ELEX 7660: Digital System Design

Infrared Communication

Abstract

The purpose of this project was to utilize infrared communication, specifically text messaging using infrared. The goal was to use the FPGA as the base to create a line of sight communication, where the FPGA takes an input from the keypad and transmit the signal through the LED, and have the signal be received through the Semiconductor photodiode connected to the FPGA, where the signal received is demodulated and displayed on the 7-segment display. The communication protocol used was the NEC protocol.

Prepared by:	Prep Date
Ben Yang	April 13, 2018
Marvin Matanguihan	April 13, 2018

Table of Contents

Table of Figures

1 Introduction

This project is about infrared communication, specifically sending and receiving text messaging using infrared through the FPGA. The main motivation behind this project is to gain a better understanding of wireless communication by exploring and implementing the simple wireless technology. The goal is to be able to apply everything learned during this process for future personal projects.

The project will take advantage of the FPGA's ability to process information in parallel for sending and receiving data at the same time. The idea is to use the FPGA as the base to create a line of sight communication system, where the FPGA takes an input and transmits it, then receives and process the signal to display information on a display.

2 Background

For this project, the infrared communication will be using the NEC protocol. This protocol can be found in many IR remote-controls and like many other IR protocol it uses pulse distance modulation. [1] Meaning that the protocol differentiates between bits 0 and 1 by the distance between each pulse. In this case, logic 0 is a 562. 5µs pulse follow by a 562.5µs of space, and logic 1 is a 562. 5µs pulse follow by a 1.6875ms of space. [2]

The format of the NEC protocol is as follows, see Figure 1:

- A 9ms pulse which indicates the start of the transmission follow by 4.5ms of space, this is known as the letter code [3]
- An 8-bit receiver device address is sent follow by the inverse of the code for redundancy
- An 8-bit command/data follow by the inverse of the logic
- A 562.5µs pulse indicating the end of the transmission

Overall, the whole transmission should take around 67.5ms; with the letter code taking 13.5ms, the address of 2 bytes taking around 27ms, the data of 2 bytes taking around 27ms and the final pulse of 562.5µs.

An important thing to keep in mind is that the bits are transmitted with the least significant bit first, and that the NEC protocol requires a 38.22kHz carrier frequency.

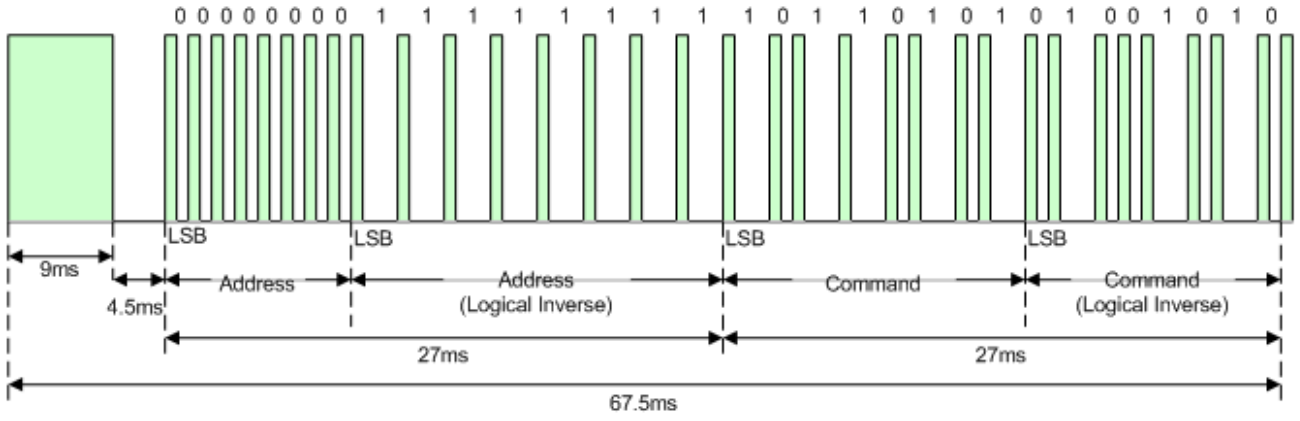

Figure 1: Example of NEC protocol[1]

[1] The diagram is from the Altium website: http://techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol

As IR transmission is not a new technology and has been around for a long time, there are many similar projects online. This project is base heavy upon two others project found online, one by Sparkfun and another by Gaurav Singh.

In the article from Sparkfun, the basics of the infrared communication is explained and an example of building one was shown. [4] The article uses an Arduino, an IR led and a IR receiver to the capture data from a TV remote controller and sent data to a TV. The IR led, and IR receiver used in this project is the same one that is present in Sparkfun's project. The parts used are a 950nm IR led, a TSOP38238 IR receiver, a keyboard for input, and a display for the output. Since the parts kit includes both the keyboard and the 7-segment display, only the led and the receiver needs to be purchased. 2 of each part were ordered to ensure there's a backup if the parts failed.

Gaurav Singh is the author of the WireFrame FPGA Board: NEC IR Receiver module with Verilog HDL. The team directly contacted Mr. Singh for permission to use his NEC decode module, and we were able to modify his code for our project. On the website[2], he describes how to decode an NEC protocol module that is received from the photodiode. As seen in Figure 2, it is shown that the signal through the photodiode is an inverted version of the transmitted signal displayed in Figure 1. Knowing that the message signal has 67 transition edges, the program would be based on the time difference between the transition edges depending on what stage. For example, if the signal transitions from a high signal to a low signal before receiving any messages, then if it stays low at around 9.5ms until its next transition edge, this could indicate that it's the start of a message. To ensure that the signal is not corrupted, the code compares each stage of the data bits received (i.e address and inverse of address, and data and inverse data).

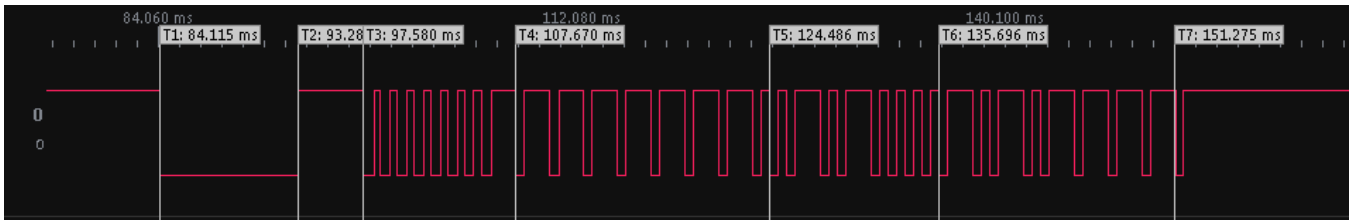

Figure 2: Example of Signal Received

[2] The website is: https://www.circuitvalley.com/2015/10/xilinx-FPGA-board-NEC-IR-receiver-decoder-VerilogHDL-VHDL-wireframe-spartan.html

3 Objectives

The objectives of the project are the following:

- Demonstrate the infrared wireless communication which uses NEC protocol
- Demonstrate the working hardware, where the input of the keypad is transmitted and displayed on the 7-segment display

To demonstrate the working system which contains all the above objectives, a signal will be generated using the keypad and transmitted through the LED. The signal transmitted will then be received by the photodiode, and data will be displayed on the 7-segment display. During this test, there will be no delay between the sending and receiving sub-system. However, in actual application, there needs a delay so there is no interference between the sub-systems.

4 High Level Project Diagrams

The following shows how the hardware components are connected to each other. There are two parts to the hardware; the transmission side on the left and the receiver side on the right. The original plan involved using the LCD display, however due to the time constraint and difficulties encounter when working with it, it was swap out with the 7-segment display

The configuration of the hardware is the setup for the demonstration where the device will transmit data to itself. However, in actual application the photo diode should configured so it is unable to see the data from the led.

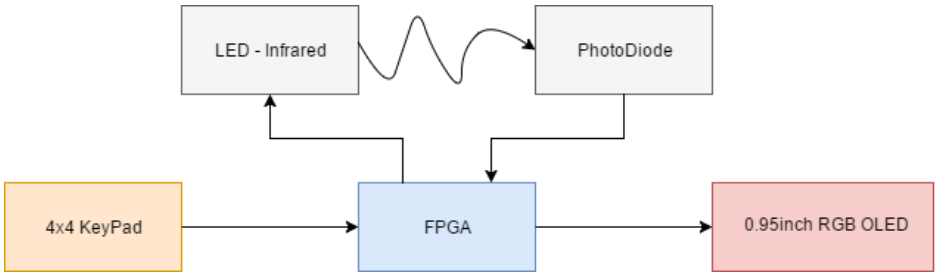

Figure 3: High Level Hardware Diagram[3]

The Figure 4 shows the inner working of the software components. The transmission side will be waiting for a key press. When it detects it, the message will be formatted for the 7-segment display, ie which led to turn on and which ones to keep off. Then, it will be sent out using the NEC protocol. The address byte will contain information on which digital the data will show at on the display, and the data will have information on the pressed key. The receiver side will be able to recognize the letter code and unpack the NEC protocol to get the address and data information. The address and data will be mapped on to the 7-segement display. At the end, the screen should show the key that was pressed.

[3] Note: the LED is not physically connected to the photodiode

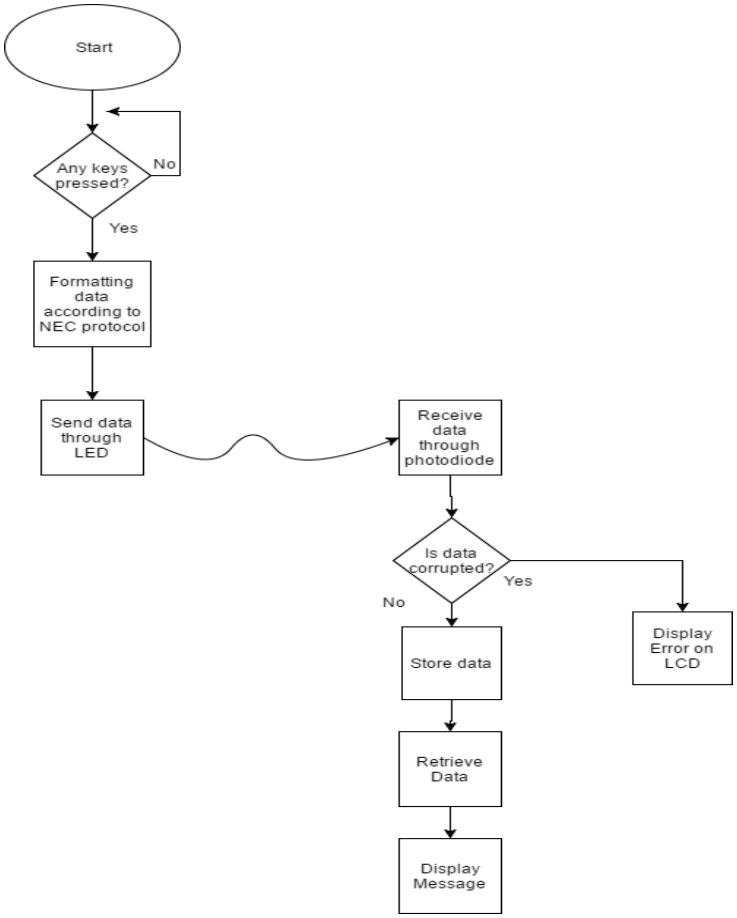

Figure 4: High Level Software Diagram

5 Results

In this part of the document, we demonstrate the test bench simulation for each module created, the demonstration of the signal transmitted and received through the oscilloscope, and the working hardware.

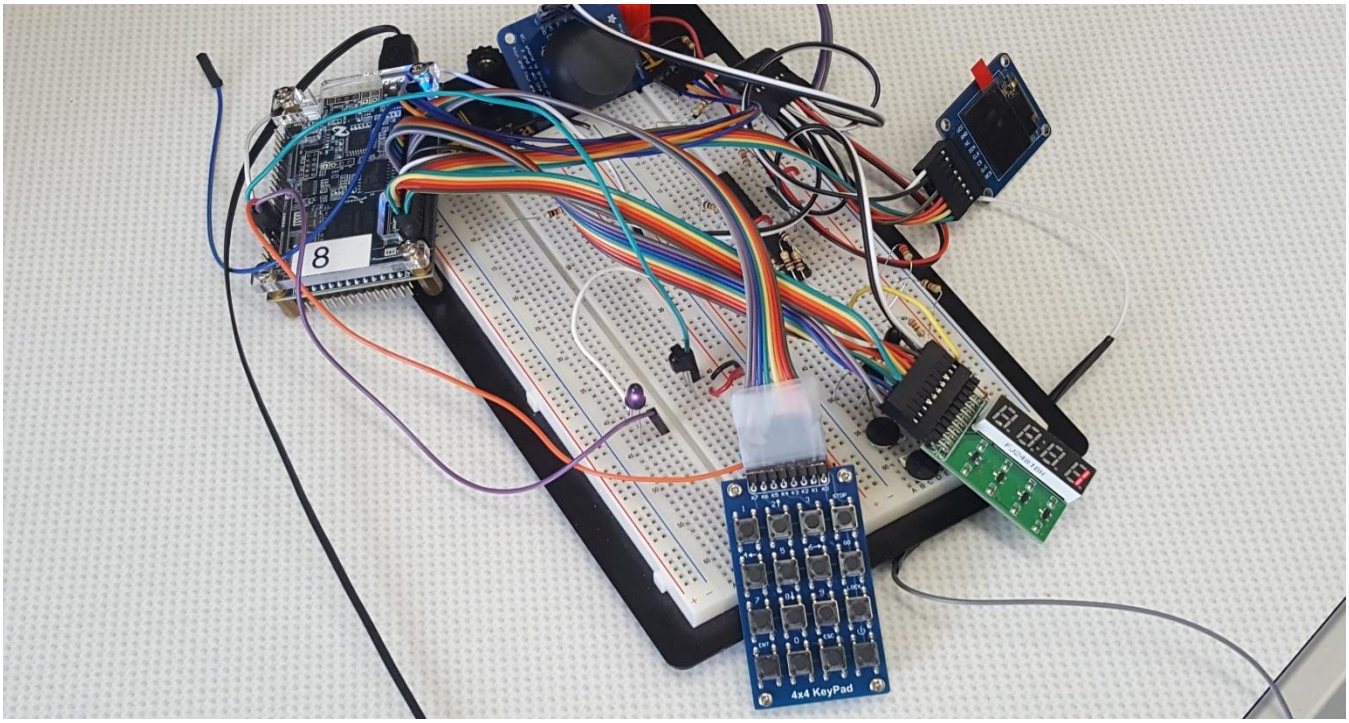

Figure 5: Layout of the Project

5.1 Test Bench Simulation

The test bench modules confirms that the signal output that each test bench is testing are correct.

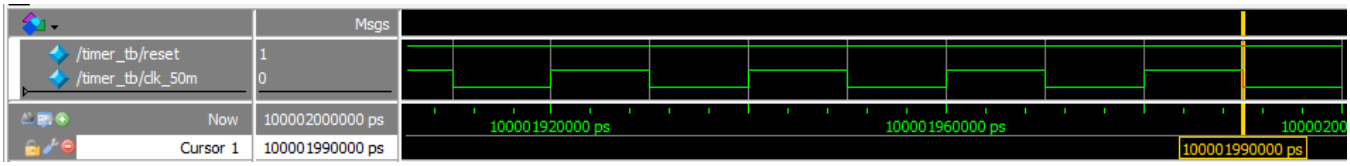

Figure 6: Simulation of Timer Module Test Bench

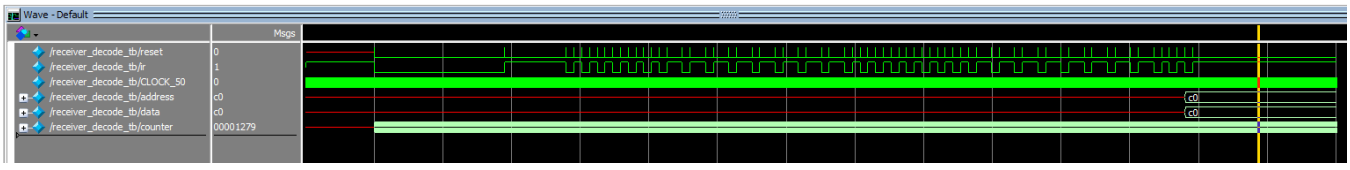

Figure 7: Simulation of the NEC Decode Module Test Bench

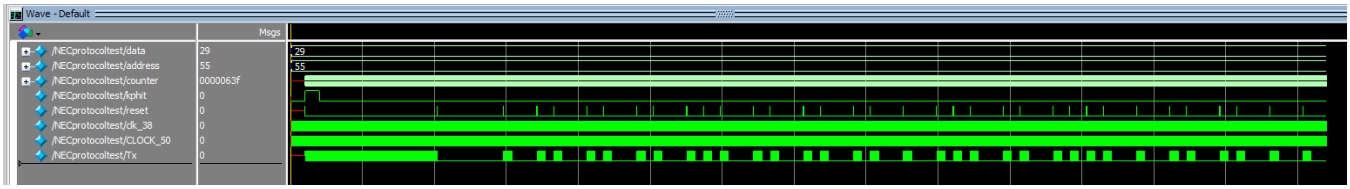

Figure 8: Simulation of the NEC Transmitter Module Test Bench

5.2 Transmitted and Received Signal

In this case, number 7 was pressed on the keypad and the output from the led shown on the left and the signal seem by the receiver is shown on the right. Figure 7 confirms that the system works as it should outputting number 7 on the 7-segment display.

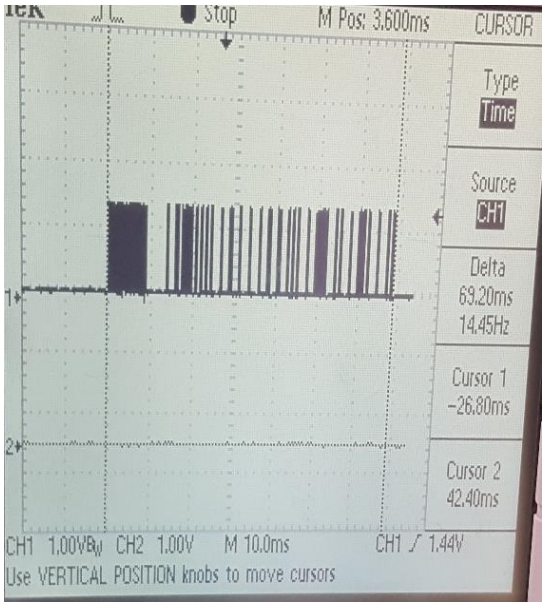

Figure 9: Transmitted Signal of 7

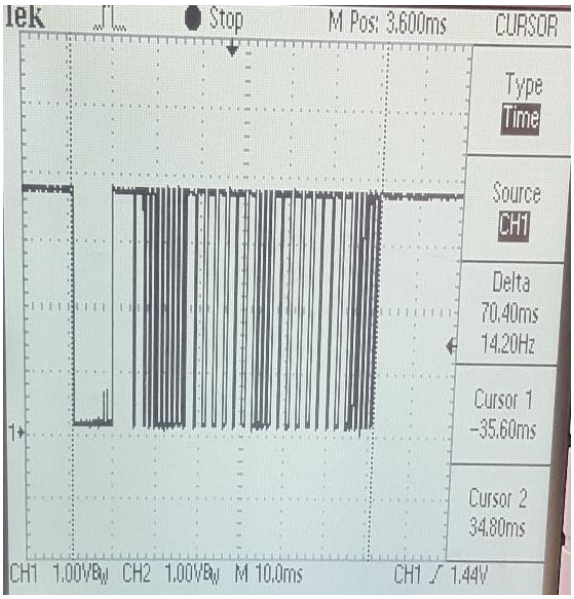

Figure 10: Received Signal of 7

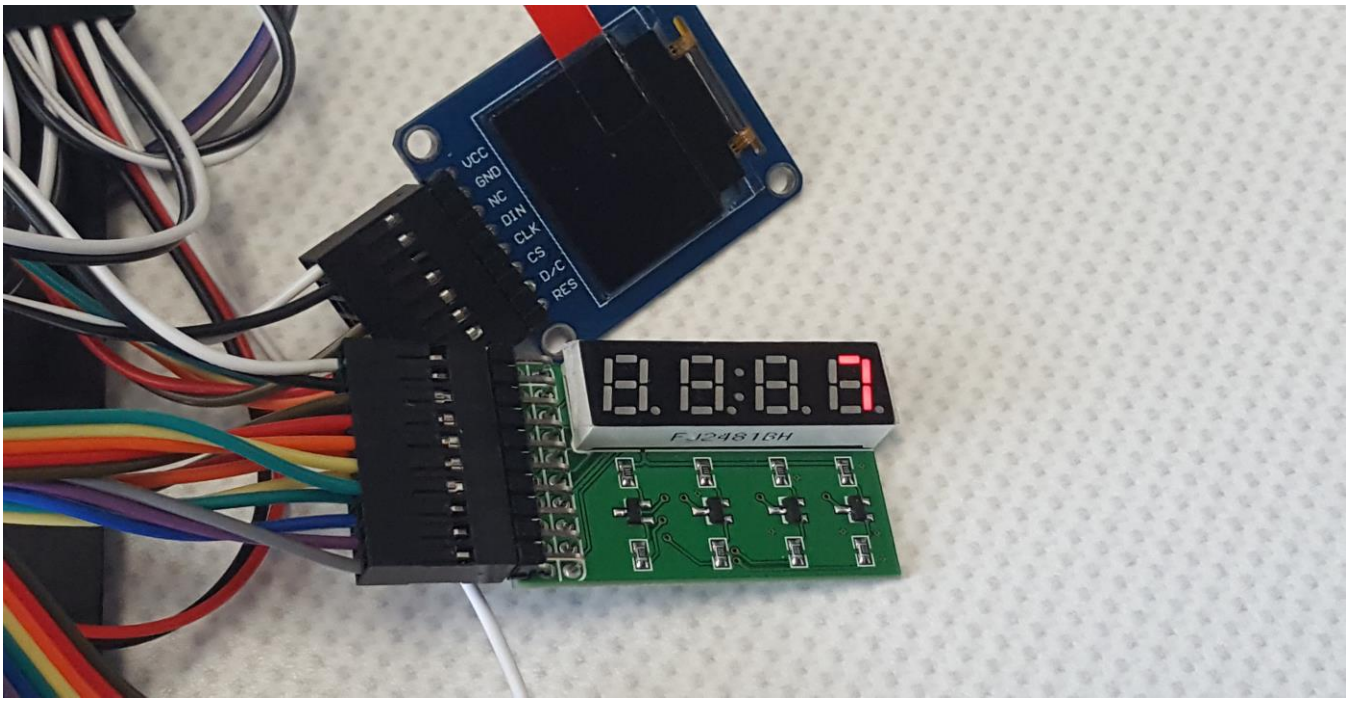

Figure 11: Displayed Number After Decoding

6 Discussion

The main purpose of the NEC protocol module is to receive the 8-bit address and data information and output it on to the led in the NEC format. The major different between this module and the others in the project is it does not use a 50MHz clock from the FPGA. Instead it uses the 76KHz from the clock_38 module, which itself was auto generated using Quartus 17.1. This is because of the 38KHz carrier frequency requirement from the NEC protocol. A pulse is actually the IR led turning on and off at 38KHz for the pulse duration. For example, outputting a 9ms pulse is the led flashing at 38KHz for 9ms, for a total of 342 times. The NEC protocol module code utilizes the timer module to keep track of the length of the pulses, while the 76KHz is used to create the 38KHz blinking. The blinking is done by incrementing the one-bit variable call flipy every positive clock cycle. Therefore, flipy is changing between 1 and 0 at the rate of 76KHz/2=38KHz.

For the NEC decoder module, it is dependent on two main components; timing and counter. The timing used in this program is the system clock of 50 MHz and the program flip-flops every positive edge of the clock. For the signal to be processed, it must undergo a transition from high to low and have a space for 9.5ms, followed up by another transition from low to high, and have the pulse width to be around 4.5ms. To keep track of those delay values, a counter is used in the timer module to increment every 1ns, then reset once the next stage has been reached.

The next stage is decoding the address and the data bits received. After going through the letter code, it is observed that the transition from high to low occurs every odd number of transitions. Utilizing this observation and with the counter resetting at every transition, we can implement a system that uses the counter value at every odd edge to determine whether the signal is a logic “1” or a logic “0”. If the pulse width is greater than 1000us, it is known that the value must be a logic “1” and else, it’s a logic “0”. That value is stored into a 32-bit buffer which contains the 8-bit address, 8-bit inverse address, 8-bit data, and 8-bit inverse data. For the signal to not be an error, it has to meet the requirement that at each stage, the counter has to be in between a certain value. Once the signal has fully been decoded, the code has to compare the address and data, and the inverse of those values that are stored in the buffer, to ensure the data is not corrupted.

7 Summary and Conclusions

The two main goals of the project were to demonstrate the infrared wireless communication using the NEC protocol, and to demonstrate that the FPGA can be used to transmit and receive signals almost simultaneously. The team was able to meet those goals and learned more on the process of transmitting and decoding the received signals. To improve upon the project, incorporating a dynamic display such as the LCD display and a complex system similar to texting would have exemplified the capabilities of the infrared wireless communication.

8 Bibliography

[1] M. Ray, "A universal algorithm for implementing an infrared decoder," EDN Network, 6 May 2009. [Online]. Available: https://www.edn.com/design/consumer/4313255/A-universal-algorithm-for-implementing-an-infrared-decoder. [Accessed 13 April 2018].

[2] "NEC Infrared Transmission Protocol," Altium, 13 Sept 2017. [Online]. Available: http://techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol. [Accessed 13 April 2018].

[3] "Data Formats for IR Remote Control," Vishay Semiconductors, 2013.

[4] "IR Communciation," Sparkfun, [Online]. Available: https://learn.sparkfun.com/tutorials/ir-communication. [Accessed 13 April 2018].

[5] G. Singh, "Circuit Valley," 12 October 2015. [Online]. [Accessed February to April 2018].

[6] G. Singh, "WireFrame FPGA Board : NEC IR Receiver module With VerilogHDL," CircuitValley, 12 October 2015. [Online]. Available: https://www.circuitvalley.com/2015/10/xilinx-FPGA-board-NEC-IR-receiver-decoder-VerilogHDL-VHDL-wireframe-spartan.html. [Accessed 13 April 2018].

A. Appendix A: Program Codes

The following section will contain all the codes written from this project.

A.1 Top Level Module

This is the top-level module where it describes how all the other modules connects together and works with the FPGA input and output.

```
// ELEX 7660 IR Project

module ProjectTop ((* altera_attribute = "-name WEAK_PULL_UP_RESISTOR ON" *)
              input logic  [3:0] kpr,  // rows, active-low w/ pull-ups
                    input logic  reset_n, CLOCK_50,
                    input logic ir, // receiving data
                    output logic [3:0] kpc,  // column select, active-low
              output logic [7:0] addressRx,   // " digit enables only use last 4
bits
                    output logic [7:0] dataRx,  // 7 seg led to turn on
                    output logic Tx // senting data
               ) ;

   logic clk ;                  // 2kHz clock for keypad scanning
   logic clk_38 ;                        // 38khz clock for NECprotocol
   logic t_resetRx, treset;
   logic [7:0] dataTx;
   logic [31:0] counterTx, counterRx;
   logic kphit ;                // a key is pressed
   logic [3:0] num ;            // value of pressed key

   assign ct = { {3{1'b0}}, kphit } ;
   pll pll0 ( .inclk0(CLOCK_50), .c0(clk) ) ;

   // instantiate your modules here...
      colseq colseq_0 ( .*);
      kpdecode kpdecode_0 ( .*);
      decode7 decode_0 ( .data(dataTx), .*);

      //clock_38 module
      clock_38k clk38(.inclk0(CLOCK_50), .c0(clk_38)) ;//  refclk.clk

      //counter
      timer timerTx (.reset(treset), .counter(counterTx), .CLOCK_50);
      timer timerRx (.reset(t_resetRx), .counter(counterRx), .CLOCK_50);

      //mod
      NECprotocol NecTx ( .data(dataTx), .address(ct), .reset(treset), .kphit,
.clk_38, .Tx, .counter(counterTx));

      //decode module
      nec_decode
NecRx(.elapsed_time(counterRx),.address(addressRx),.data(dataRx),.reset(t_resetRx),
.* ) ;
endmodule



// megafunction wizard: %ALTPLL%
// ...
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
```

```
// ...

module pll ( inclk0, c0);

        input     inclk0;
        output    c0;

        wire [0:0] sub_wire2 = 1'h0;
        wire [4:0] sub_wire3;
        wire  sub_wire0 = inclk0;
        wire [1:0] sub_wire1 = {sub_wire2, sub_wire0};
        wire [0:0] sub_wire4 = sub_wire3[0:0];
        wire  c0 = sub_wire4;

        altpll altpll_component ( .inclk (sub_wire1), .clk
          (sub_wire3), .activeclock (), .areset (1'b0), .clkbad
          (), .clkena ({6{1'b1}}), .clkloss (), .clkswitch
          (1'b0), .configupdate (1'b0), .enable0 (), .enable1 (),
          .extclk (), .extclkena ({4{1'b1}}), .fbin (1'b1),
          .fbmimicbidir (), .fbout (), .fref (), .icdrclk (),
          .locked (), .pfdena (1'b1), .phasecounterselect
          ({4{1'b1}}), .phasedone (), .phasestep (1'b1),
          .phaseupdown (1'b1), .pllena (1'b1), .scanaclr (1'b0),
          .scanclk (1'b0), .scanclkena (1'b1), .scandata (1'b0),
          .scandataout (), .scandone (), .scanread (1'b0),
          .scanwrite (1'b0), .sclkout0 (), .sclkout1 (),
          .vcooverrange (), .vcounderrange ());

        defparam
                altpll_component.bandwidth_type = "AUTO",
                altpll_component.clk0_divide_by = 25000,
                altpll_component.clk0_duty_cycle = 50,
                altpll_component.clk0_multiply_by = 1,
                altpll_component.clk0_phase_shift = "0",
                altpll_component.compensate_clock = "CLK0",
                altpll_component.inclk0_input_frequency = 20000,
                altpll_component.intended_device_family = "Cyclone IV E",
                altpll_component.lpm_hint = "CBX_MODULE_PREFIX=lab1clk",
                altpll_component.lpm_type = "altpll",
                altpll_component.operation_mode = "NORMAL",
                altpll_component.pll_type = "AUTO",
                altpll_component.port_activeclock = "PORT_UNUSED",
                altpll_component.port_areset = "PORT_UNUSED",
                altpll_component.port_clkbad0 = "PORT_UNUSED",
                altpll_component.port_clkbad1 = "PORT_UNUSED",
                altpll_component.port_clkloss = "PORT_UNUSED",
                altpll_component.port_clkswitch = "PORT_UNUSED",
                altpll_component.port_configupdate = "PORT_UNUSED",
                altpll_component.port_fbin = "PORT_UNUSED",
                altpll_component.port_inclk0 = "PORT_USED",
                altpll_component.port_inclk1 = "PORT_UNUSED",
                altpll_component.port_locked = "PORT_UNUSED",
                altpll_component.port_pfdena = "PORT_UNUSED",
                altpll_component.port_phasecounterselect = "PORT_UNUSED",
                altpll_component.port_phasedone = "PORT_UNUSED",
                altpll_component.port_phasestep = "PORT_UNUSED",
                altpll_component.port_phaseupdown = "PORT_UNUSED",
                altpll_component.port_pllena = "PORT_UNUSED",
                altpll_component.port_scanaclr = "PORT_UNUSED",
                altpll_component.port_scanclk = "PORT_UNUSED",
```

```
                altpll_component.port_scanclkena = "PORT_UNUSED",
                altpll_component.port_scandata = "PORT_UNUSED",
                altpll_component.port_scandataout = "PORT_UNUSED",
                altpll_component.port_scandone = "PORT_UNUSED",
                altpll_component.port_scanread = "PORT_UNUSED",
                altpll_component.port_scanwrite = "PORT_UNUSED",
                altpll_component.port_clk0 = "PORT_USED",
                altpll_component.port_clk1 = "PORT_UNUSED",
                altpll_component.port_clk2 = "PORT_UNUSED",
                altpll_component.port_clk3 = "PORT_UNUSED",
                altpll_component.port_clk4 = "PORT_UNUSED",
                altpll_component.port_clk5 = "PORT_UNUSED",
                altpll_component.port_clkena0 = "PORT_UNUSED",
                altpll_component.port_clkena1 = "PORT_UNUSED",
                altpll_component.port_clkena2 = "PORT_UNUSED",
                altpll_component.port_clkena3 = "PORT_UNUSED",
                altpll_component.port_clkena4 = "PORT_UNUSED",
                altpll_component.port_clkena5 = "PORT_UNUSED",
                altpll_component.port_extclk0 = "PORT_UNUSED",
                altpll_component.port_extclk1 = "PORT_UNUSED",
                altpll_component.port_extclk2 = "PORT_UNUSED",
                altpll_component.port_extclk3 = "PORT_UNUSED",
                altpll_component.width_clock = 5;

endmodule
```

A.2 colseq.sv

Here is the colseq module that accepts the input from the keypad from row and outputs the column that the key was pressed.

```
//Drives the keypad
//Input: keypad rows(kpr), clock(clk), active-low reset(reset_n)
//Output: 4-bit output driving the keypad columns(kpc)

module colseq (input logic clk, reset_n,
                              input logic [3:0] kpr,
                              output logic [3:0] kpc);

      logic [3:0] kpcnext;

      always_comb begin
            if (~reset_n)
                  kpcnext = 4'b0111;
            else if (kpr == 4'b1111)
            begin
                  case(kpc)
                        4'b0111: kpcnext = 4'b1011;
                        4'b1011: kpcnext = 4'b1101;
                        4'b1101: kpcnext = 4'b1110;
                        4'b1110: kpcnext = 4'b0111;
                  default: kpcnext = 4'b0111;
                  endcase
            end
            else
                  kpcnext = 4'b0111;
      end

      always_ff @(posedge clk) begin
            kpc <= kpcnext;
```

```
    end
endmodule
```

A.3 kpdecode.sv

Here is the kpdecode module that accepts an input from the keypad (kpc and kpr) and outputs that a key has been pressed (kphit) and the number pressed (num).

```
//kpdecode.sv - Accepts a 4-bit input kpr, input kcr, and outputs 4-bit logic kphit
and logic kphit to indicate button was pressed

module kpdecode(input logic [3:0] kpr, kpc,
                                    output logic [3:0] num,
                                          output logic kphit);


    //next-state logic
    always_comb begin
          num = 4'b00;
          if (kpr != 4'b1111) begin           //if one of the keys are pressed
                kphit = 1;
                if (kpr == 4'b0111) begin     //Row 1
                      if (kpc == 4'b0111)     //Col 1
                            num = 4'h1;
                      if (kpc == 4'b1011)     //Col 2
                            num = 4'h2;
                      if (kpc == 4'b1101)     //Col 3
                            num = 4'h3;
                      if (kpc == 4'b1110)     //Col 4
                            num = 4'ha;
                end
                else if (kpr == 4'b1011) begin      //Row 2
                      if (kpc == 4'b0111)     //Col 1
                            num = 4'h4;
                      if (kpc == 4'b1011)     //Col 2
                            num = 4'h5;
                      if (kpc == 4'b1101)     //Col 3
                            num = 4'h6;
                      if (kpc == 4'b1110)     //Col 4
                            num = 4'hb;
                end
                else if (kpr == 4'b1101) begin      //Row 3
                      if (kpc == 4'b0111)     //Col 1
                            num = 4'h7;
                      if (kpc == 4'b1011)     //Col 2
                            num = 4'h8;
                      if (kpc == 4'b1101)     //Col 3
                            num = 4'h9;
                      if (kpc == 4'b1110)     //Col 4
                            num = 4'hc;
                end
                else if (kpr == 4'b1110) begin      //Row 4
                      if (kpc == 4'b0111)     //Col 1
                            num = 4'he;
                      if (kpc == 4'b1011)     //Col 2
                            num = 4'h0;
                      if (kpc == 4'b1101)     //Col 3
                            num = 4'hf;
                      if (kpc == 4'b1110)     //Col 4
                            num = 4'hd;
```

```
            end
        end
        else
        begin
            kphit = 0;
        end


    end
endmodule
```

A.4 clock38.sv

The module clock_38 is used to transmit signal at 38 KHz, since the infrared communication requires that.

```
// megafunction wizard: %ALTPLL%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altpll

// ============================================================
// File Name: clock_38.v
// Megafunction Name(s):
//                altpll
//
// Simulation Library Files(s):
//                altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 17.1.0 Build 590 10/25/2017 SJ Lite Edition
// ************************************************************


//Copyright (C) 2017  Intel Corporation. All rights reserved.
//Your use of Intel Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License Agreement,
//the Intel FPGA IP License Agreement, or other applicable license
//agreement, including, without limitation, that your use is for
//the sole purpose of programming logic devices manufactured by
//Intel and sold by Intel or its authorized distributors.  Please
//refer to the applicable agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module clock_38 (
      areset,
      inclk0,
      c0);

      input   areset;
```

```
	input	inclk0;
	output		c0;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
	tri0	areset;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

	wire [0:0] sub_wire2 = 1'h0;
	wire [4:0] sub_wire3;
	wire  sub_wire0 = inclk0;
	wire [1:0] sub_wire1 = {sub_wire2, sub_wire0};
	wire [0:0] sub_wire4 = sub_wire3[0:0];
	wire  c0 = sub_wire4;

	altpll	altpll_component (
				.areset (areset),
				.inclk (sub_wire1),
				.clk (sub_wire3),
				.activeclock (),
				.clkbad (),
				.clkena ({6{1'b1}}),
				.clkloss (),
				.clkswitch (1'b0),
				.configupdate (1'b0),
				.enable0 (),
				.enable1 (),
				.extclk (),
				.extclkena ({4{1'b1}}),
				.fbin (1'b1),
				.fbmimicbidir (),
				.fbout (),
				.fref (),
				.icdrclk (),
				.locked (),
				.pfdena (1'b1),
				.phasecounterselect ({4{1'b1}}),
				.phasedone (),
				.phasestep (1'b1),
				.phaseupdown (1'b1),
				.pllena (1'b1),
				.scanaclr (1'b0),
				.scanclk (1'b0),
				.scanclkena (1'b1),
				.scandata (1'b0),
				.scandataout (),
				.scandone (),
				.scanread (1'b0),
				.scanwrite (1'b0),
				.sclkout0 (),
				.sclkout1 (),
				.vcooverrange (),
				.vcounderrange ());
	defparam
		altpll_component.bandwidth_type = "AUTO",
		altpll_component.clk0_divide_by = 6250,
		altpll_component.clk0_duty_cycle = 50,
		altpll_component.clk0_multiply_by = 19,
```

```
        altpll_component.clk0_phase_shift = "0",
        altpll_component.compensate_clock = "CLK0",
        altpll_component.inclk0_input_frequency = 20000,
        altpll_component.intended_device_family = "Cyclone IV E",
        altpll_component.lpm_hint = "CBX_MODULE_PREFIX=clock_38",
        altpll_component.lpm_type = "altpll",
        altpll_component.operation_mode = "NORMAL",
        altpll_component.pll_type = "AUTO",
        altpll_component.port_activeclock = "PORT_UNUSED",
        altpll_component.port_areset = "PORT_USED",
        altpll_component.port_clkbad0 = "PORT_UNUSED",
        altpll_component.port_clkbad1 = "PORT_UNUSED",
        altpll_component.port_clkloss = "PORT_UNUSED",
        altpll_component.port_clkswitch = "PORT_UNUSED",
        altpll_component.port_configupdate = "PORT_UNUSED",
        altpll_component.port_fbin = "PORT_UNUSED",
        altpll_component.port_inclk0 = "PORT_USED",
        altpll_component.port_inclk1 = "PORT_UNUSED",
        altpll_component.port_locked = "PORT_UNUSED",
        altpll_component.port_pfdena = "PORT_UNUSED",
        altpll_component.port_phasecounterselect = "PORT_UNUSED",
        altpll_component.port_phasedone = "PORT_UNUSED",
        altpll_component.port_phasestep = "PORT_UNUSED",
        altpll_component.port_phaseupdown = "PORT_UNUSED",
        altpll_component.port_pllena = "PORT_UNUSED",
        altpll_component.port_scanaclr = "PORT_UNUSED",
        altpll_component.port_scanclk = "PORT_UNUSED",
        altpll_component.port_scanclkena = "PORT_UNUSED",
        altpll_component.port_scandata = "PORT_UNUSED",
        altpll_component.port_scandataout = "PORT_UNUSED",
        altpll_component.port_scandone = "PORT_UNUSED",
        altpll_component.port_scanread = "PORT_UNUSED",
        altpll_component.port_scanwrite = "PORT_UNUSED",
        altpll_component.port_clk0 = "PORT_USED",
        altpll_component.port_clk1 = "PORT_UNUSED",
        altpll_component.port_clk2 = "PORT_UNUSED",
        altpll_component.port_clk3 = "PORT_UNUSED",
        altpll_component.port_clk4 = "PORT_UNUSED",
        altpll_component.port_clk5 = "PORT_UNUSED",
        altpll_component.port_clkena0 = "PORT_UNUSED",
        altpll_component.port_clkena1 = "PORT_UNUSED",
        altpll_component.port_clkena2 = "PORT_UNUSED",
        altpll_component.port_clkena3 = "PORT_UNUSED",
        altpll_component.port_clkena4 = "PORT_UNUSED",
        altpll_component.port_clkena5 = "PORT_UNUSED",
        altpll_component.port_extclk0 = "PORT_UNUSED",
        altpll_component.port_extclk1 = "PORT_UNUSED",
        altpll_component.port_extclk2 = "PORT_UNUSED",
        altpll_component.port_extclk3 = "PORT_UNUSED",
        altpll_component.width_clock = 5;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ACTIVECLK_CHECK STRING "0"
// Retrieval info: PRIVATE: BANDWIDTH STRING "1.000"
// Retrieval info: PRIVATE: BANDWIDTH_FEATURE_ENABLED STRING "1"
```

```
// Retrieval info: PRIVATE: BANDWIDTH_FREQ_UNIT STRING "MHz"
// Retrieval info: PRIVATE: BANDWIDTH_PRESET STRING "Low"
// Retrieval info: PRIVATE: BANDWIDTH_USE_AUTO STRING "1"
// Retrieval info: PRIVATE: BANDWIDTH_USE_PRESET STRING "0"
// Retrieval info: PRIVATE: CLKBAD_SWITCHOVER_CHECK STRING "0"
// Retrieval info: PRIVATE: CLKLOSS_CHECK STRING "0"
// Retrieval info: PRIVATE: CLKSWITCH_CHECK STRING "0"
// Retrieval info: PRIVATE: CNX_NO_COMPENSATE_RADIO STRING "0"
// Retrieval info: PRIVATE: CREATE_CLKBAD_CHECK STRING "0"
// Retrieval info: PRIVATE: CREATE_INCLK1_CHECK STRING "0"
// Retrieval info: PRIVATE: CUR_DEDICATED_CLK STRING "c0"
// Retrieval info: PRIVATE: CUR_FBIN_CLK STRING "c0"
// Retrieval info: PRIVATE: DEVICE_SPEED_GRADE STRING "Any"
// Retrieval info: PRIVATE: DIV_FACTOR0 NUMERIC "1"
// Retrieval info: PRIVATE: DUTY_CYCLE0 STRING "50.00000000"
// Retrieval info: PRIVATE: EFF_OUTPUT_FREQ_VALUE0 STRING "0.152000"
// Retrieval info: PRIVATE: EXPLICIT_SWITCHOVER_COUNTER STRING "0"
// Retrieval info: PRIVATE: EXT_FEEDBACK_RADIO STRING "0"
// Retrieval info: PRIVATE: GLOCKED_COUNTER_EDIT_CHANGED STRING "1"
// Retrieval info: PRIVATE: GLOCKED_FEATURE_ENABLED STRING "0"
// Retrieval info: PRIVATE: GLOCKED_MODE_CHECK STRING "0"
// Retrieval info: PRIVATE: GLOCK_COUNTER_EDIT NUMERIC "1048575"
// Retrieval info: PRIVATE: HAS_MANUAL_SWITCHOVER STRING "1"
// Retrieval info: PRIVATE: INCLK0_FREQ_EDIT STRING "50.000"
// Retrieval info: PRIVATE: INCLK0_FREQ_UNIT_COMBO STRING "MHz"
// Retrieval info: PRIVATE: INCLK1_FREQ_EDIT STRING "100.000"
// Retrieval info: PRIVATE: INCLK1_FREQ_EDIT_CHANGED STRING "1"
// Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_CHANGED STRING "1"
// Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_COMBO STRING "MHz"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
// Retrieval info: PRIVATE: INT_FEEDBACK__MODE_RADIO STRING "1"
// Retrieval info: PRIVATE: LOCKED_OUTPUT_CHECK STRING "0"
// Retrieval info: PRIVATE: LONG_SCAN_RADIO STRING "1"
// Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE STRING "Not Available"
// Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE_DIRTY NUMERIC "0"
// Retrieval info: PRIVATE: LVDS_PHASE_SHIFT_UNIT0 STRING "deg"
// Retrieval info: PRIVATE: MIG_DEVICE_SPEED_GRADE STRING "Any"
// Retrieval info: PRIVATE: MIRROR_CLK0 STRING "0"
// Retrieval info: PRIVATE: MULT_FACTOR0 NUMERIC "1"
// Retrieval info: PRIVATE: NORMAL_MODE_RADIO STRING "1"
// Retrieval info: PRIVATE: OUTPUT_FREQ0 STRING "0.15200000"
// Retrieval info: PRIVATE: OUTPUT_FREQ_MODE0 STRING "1"
// Retrieval info: PRIVATE: OUTPUT_FREQ_UNIT0 STRING "MHz"
// Retrieval info: PRIVATE: PHASE_RECONFIG_FEATURE_ENABLED STRING "1"
// Retrieval info: PRIVATE: PHASE_RECONFIG_INPUTS_CHECK STRING "0"
// Retrieval info: PRIVATE: PHASE_SHIFT0 STRING "0.00000000"
// Retrieval info: PRIVATE: PHASE_SHIFT_STEP_ENABLED_CHECK STRING "0"
// Retrieval info: PRIVATE: PHASE_SHIFT_UNIT0 STRING "deg"
// Retrieval info: PRIVATE: PLL_ADVANCED_PARAM_CHECK STRING "0"
// Retrieval info: PRIVATE: PLL_ARESET_CHECK STRING "1"
// Retrieval info: PRIVATE: PLL_AUTOPLL_CHECK NUMERIC "1"
// Retrieval info: PRIVATE: PLL_ENHPLL_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: PLL_FASTPLL_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: PLL_FBMIMIC_CHECK STRING "0"
// Retrieval info: PRIVATE: PLL_LVDS_PLL_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: PLL_PFDENA_CHECK STRING "0"
// Retrieval info: PRIVATE: PLL_TARGET_HARCOPY_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: PRIMARY_CLK_COMBO STRING "inclk0"
// Retrieval info: PRIVATE: RECONFIG_FILE STRING "clock_38.mif"
// Retrieval info: PRIVATE: SACN_INPUTS_CHECK STRING "0"
```

```
// Retrieval info: PRIVATE: SCAN_FEATURE_ENABLED STRING "1"
// Retrieval info: PRIVATE: SELF_RESET_LOCK_LOSS STRING "0"
// Retrieval info: PRIVATE: SHORT_SCAN_RADIO STRING "0"
// Retrieval info: PRIVATE: SPREAD_FEATURE_ENABLED STRING "0"
// Retrieval info: PRIVATE: SPREAD_FREQ STRING "50.000"
// Retrieval info: PRIVATE: SPREAD_FREQ_UNIT STRING "KHz"
// Retrieval info: PRIVATE: SPREAD_PERCENT STRING "0.500"
// Retrieval info: PRIVATE: SPREAD_USE STRING "0"
// Retrieval info: PRIVATE: SRC_SYNCH_COMP_RADIO STRING "0"
// Retrieval info: PRIVATE: STICKY_CLK0 STRING "1"
// Retrieval info: PRIVATE: SWITCHOVER_COUNT_EDIT NUMERIC "1"
// Retrieval info: PRIVATE: SWITCHOVER_FEATURE_ENABLED STRING "1"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: USE_CLK0 STRING "1"
// Retrieval info: PRIVATE: USE_CLKENA0 STRING "0"
// Retrieval info: PRIVATE: USE_MIL_SPEED_GRADE NUMERIC "0"
// Retrieval info: PRIVATE: ZERO_DELAY_RADIO STRING "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: BANDWIDTH_TYPE STRING "AUTO"
// Retrieval info: CONSTANT: CLK0_DIVIDE_BY NUMERIC "6250"
// Retrieval info: CONSTANT: CLK0_DUTY_CYCLE NUMERIC "50"
// Retrieval info: CONSTANT: CLK0_MULTIPLY_BY NUMERIC "19"
// Retrieval info: CONSTANT: CLK0_PHASE_SHIFT STRING "0"
// Retrieval info: CONSTANT: COMPENSATE_CLOCK STRING "CLK0"
// Retrieval info: CONSTANT: INCLK0_INPUT_FREQUENCY NUMERIC "20000"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altpll"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "NORMAL"
// Retrieval info: CONSTANT: PLL_TYPE STRING "AUTO"
// Retrieval info: CONSTANT: PORT_ACTIVECLOCK STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_ARESET STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_CLKBAD0 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CLKBAD1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CLKLOSS STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CLKSWITCH STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CONFIGUPDATE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_FBIN STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_INCLK0 STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_INCLK1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_LOCKED STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PFDENA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASECOUNTERSELECT STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASEDONE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASESTEP STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASEUPDOWN STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PLLENA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANACLR STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANCLK STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANCLKENA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANDATA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANDATAOUT STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANDONE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANREAD STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANWRITE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk0 STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_clk1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk2 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk3 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk4 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk5 STRING "PORT_UNUSED"
```

```
// Retrieval info: CONSTANT: PORT_clkena0 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena2 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena3 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena4 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena5 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk0 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk2 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk3 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: WIDTH_CLOCK NUMERIC "5"
// Retrieval info: USED_PORT: @clk 0 0 5 0 OUTPUT_CLK_EXT VCC "@clk[4..0]"
// Retrieval info: USED_PORT: areset 0 0 0 0 INPUT GND "areset"
// Retrieval info: USED_PORT: c0 0 0 0 0 OUTPUT_CLK_EXT VCC "c0"
// Retrieval info: USED_PORT: inclk0 0 0 0 0 INPUT_CLK_EXT GND "inclk0"
// Retrieval info: CONNECT: @areset 0 0 0 0 areset 0 0 0 0
// Retrieval info: CONNECT: @inclk 0 0 1 1 GND 0 0 0 0
// Retrieval info: CONNECT: @inclk 0 0 1 0 inclk0 0 0 0 0
// Retrieval info: CONNECT: c0 0 0 0 0 @clk 0 0 1 0
// Retrieval info: GEN_FILE: TYPE_NORMAL clock_38.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL clock_38.ppf TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL clock_38.inc TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL clock_38.cmp TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL clock_38.bsf TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL clock_38_inst.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL clock_38_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf
// Retrieval info: CBX_MODULE_PREFIX: ON
```

A.5 NECprotocol.sv

The module NECprotocol outputs the address and the key pressed that will be transmitted through the LED.

```
//Create a NEC Infrared Transmission Protocol
//Timing:
//    Letter Code: 9ms burst + 4.5ms space = 13.5ms
//    logic 0: 562.5us burst + 562.5us space = 1.125ms
//    logic 1: 562.5us burst + 1.6875ms space = 2.25ms
//NEC Layout:
//    Letter Code(13.5ms) + (8bit address + 8bit ~address)(27ms) + (8bit data +
8bit ~data)(27ms) = 67.5ms
//Carrier Frequency of 38kHz

module NECprotocol ( input logic [7:0] data, address,
            input logic [31:0] counter, //timer count in 1us
            input logic kphit, clk_38,
            output logic reset, Tx); //turning Infrared diode on/off

      //timing parameter in us
      parameter [15:0] Tick9 = 8995; //9ms
      parameter [15:0] Tick4o5 = 4495; //4.5ms
      parameter [15:0] Ticko562 = 560; //562us
      parameter [15:0] Tick1o688 = 1683; //1.688ms
      parameter [15:0] Tick1o125 = 1120; //logic 0 length
      parameter [15:0] Tick2o25 = 2245; //logic 1 length
      parameter [15:0] Tick500 = 500000; //leter code length

      logic flipy = 0;
      logic [7:0] trackp = 0;
```

```
logic ihatemyself = 0;
logic [7:0] addressbuff, databuff;

always_ff @(posedge clk_38)
begin

	if((trackp==0)&&kphit)
	begin
		addressbuff <= address;
		databuff <= data;
		reset = 1;
		trackp <= trackp + 1;
	end
	else
	begin
		//letter head
		if(trackp == 1)
		begin
			reset = 1'b0;
			if(counter<Tick9)
			begin
				Tx <= flipy?1:0;
			end
			else
			begin
				reset = 1'b1 ;
				trackp <= trackp + 1;
			end
		end
		if(trackp == 2)
		begin
			reset = 1'b0 ;
			if(counter<Tick4o5)
				Tx <= 0;
			else
			begin
				reset = 1'b1;
				trackp <= trackp + 1;
			end
		end

		//address
		if((trackp>=3)&&(trackp<11))
		begin
			reset = 1'b0 ;
			if(counter<Ticko562)
				Tx <= flipy?1:0;
			else
			begin
				Tx <= 0;
				if(addressbuff[trackp-3])
				begin
					if(counter>=Tick2o25)
					begin
						reset = 1;
						trackp <= trackp + 1;
					end
				end
				else
				begin
```

```
                        if(counter>=Tick1o125)
                        begin
                                reset = 1;
                                trackp <= trackp + 1;
                        end
                end
        end
        //~address
        if((trackp>=11)&&(trackp<19))
        begin
                reset = 1'b0 ;
                if(counter<Ticko562)
                        Tx <= flipy?1:0;
                else
                begin
                        Tx <= 0;
                        if(~addressbuff[trackp-11])
                        begin
                                if(counter>=Tick2o25)
                                begin
                                        reset = 1;
                                        trackp <= trackp + 1;
                                end
                        end
                        else
                        begin
                                if(counter>=Tick1o125)
                                begin
                                        reset = 1;
                                        trackp <= trackp + 1;
                                end
                        end
                end
        end
        //data
        if((trackp>=19)&&(trackp<27))
        begin
                reset = 1'b0 ;
                if(counter<Ticko562)
                        Tx <= flipy?1:0;
                else
                begin
                        Tx <= 0;
                        if(databuff[trackp-19])
                        begin
                                if(counter>=Tick2o25)
                                begin
                                        reset = 1;
                                        trackp <= trackp + 1;
                                end
                        end
                        else
                        begin
                                if(counter>=Tick1o125)
                                begin
                                        reset = 1;
                                        trackp <= trackp + 1;
                                end
                        end
```

```
                        end
                    end
                    //~data
                    if((trackp>=27)&&(trackp<35))
                    begin
                        reset = 1'b0 ;
                        if(counter<Ticko562)
                            Tx <= flipy?1:0;
                        else
                        begin
                            Tx <= 0;
                            if(~databuff[trackp-27])
                            begin
                                if(counter>=Tick2o25)
                                begin
                                    reset = 1;
                                    trackp <= trackp + 1;
                                end
                            end
                            else
                            begin
                                if(counter>=Tick1o125)
                                begin
                                    reset = 1;
                                    trackp <= trackp + 1;
                                end
                            end
                        end
                    end

                    //final burst
                    if(trackp == 35)
                    begin
                        reset = 1'b0;
                        if(counter<Ticko562)
                            Tx <= flipy?1:0;
                        else
                        begin
                            if(counter>Tick500)
                                trackp <= 0;
                        end
                    end

                end
                flipy++;
            end

endmodule
```

A.6 nec_decode.sv

The nec_decode module decodes the signal received through the photodiode and extracts the address and the data from the message.

```
//accepts the clk and the IR signal as inputs, and outputs the decode num
//March 30, 2018
//

/*
```

```
necpoj == 1       we just detected the first edge of the input singal it may also
mean(if interrupt is not false) that the 9ms leading pulse started
                           after the first edge THE next pulse is expected to arrive
around 9ms so the TIMEOUT is set to 11ms and PREPULSE is set to 8ms


necpoj == 2         we just detected the second edge of the input signal and we
finished the 9ms leding pulse and now 4.5ms space started
                           after the second edge the next pulse is expected to
arrive around 4.5ms so TIMEOUT is set to 5.5ms and PREPULSE is 3ms


necpoj == 3         we just detected the third edge of the input singal and we
finished 4.5ms space and addres lsb is now started
                           after the third edge the next pulse is expected to arrive
around 562.5us so TIMEOUT is set to 2.3ms and PREPULSE is 0.2ms (timeout can be
much less at this state but to do this i have to add one more if else statemetnt)

necpoj == 4         we just detected the forth edge and the 562.5 us burt of LSB of
address has ended now a little space for '0'562.5us  or for '1' 1.6875ms
                           after the forth edge the next pulse is expected to arrive
for '0' around 562.5us  and for '1' 1.675ms so TIMEOUT is set to 2.3ms and PREPULSE
is 0.2ms

necpoj ==5 to 66  data pulse keep comming
                              TIMOUT and PREPLUSE remain same as above.

necpoj ==67         we just fined the command inverse MSB space not the final
562.5us burst has stated  so we fined the receiveing
                           now we will check the address and command for being
correct
*///H:/Personal Folders/Project/decode_tb/nec_decode.sv

module nec_decode(input logic CLOCK_50, ir,
                              input logic [31:0] elapsed_time,
                              output logic reset,
                              output logic [7:0] address, data
                              /*output logic [31:0] elapsed_out*/);

      //Constants - TICKS are in ms
      //state 1 - leading pulse has started
      parameter TICK11ms = 32'd11000 ;    //the pulse should occur before this time
for stage 1
      parameter TICK8ms = 32'd8000 ;            //the pulse

      //state 2 - 4.5ms space has started
      parameter TICK5o5ms = 32'd5500 ;
      parameter TICK3ms =  32'd3000 ;

      //stage 3 - lsb address has started
      parameter TICK2o3ms = 32'd2300 ;
      parameter TICK0o2ms = 32'd200 ;

      //stage 4 - checking whether data is logic 1 or logic 0
      parameter TICK1o0ms = 32'd800; //if greater than this number - logic 1, if
less - logic 0

      //Variables
      logic [31:0] necpoj = 1'b0 ;  //(necpoj =  NEC position) this variable is
used to keep track of edges of the input signal
```

```
      logic [31:0] timeout = TICK11ms ; //this variable is used to check whether
the pulse has exceeded this time. If it has, error
      logic [31:0] prepulse = TICK8ms ; //this variable is used to check whether
the pulse has occured before this time. If it has, error
      logic [31:0] rxbuffer;  //contains 8 bit address, 8 bit inverse address, 8
bit data, 8 bit inverse data
      logic [7:0] data_buff, address_buff ;
      logic [1:0] last , last_data = 2'b11;

      always_ff@( posedge CLOCK_50 ) begin

            if( reset ) begin
                  reset = 1'b0 ;
            end

            if (last[0] ^ last[1] ) begin

                  // Check elapsed time at each stage
                  if ( ( elapsed_time > prepulse ) && ( elapsed_time < timeout ) )
begin

                        //9ms burst has ended
                        if ( ( necpoj == 1 ) || ( necpoj == 2 ) ) begin
                              if ( ( necpoj == 1 ) && ir ) begin  //stage 2 - 4.5
ms has started
                                    necpoj = necpoj + 1'b1 ;
                                    prepulse = TICK3ms ;    // 3ms pulse
                                    timeout = TICK5o5ms ;   // 5ms pulse
                              end
                              else if ( ( !ir ) && ( necpoj == 2 ) ) begin //stage
3 - lsb of the address
                                    necpoj = necpoj + 1'b1 ;
                                    timeout = TICK2o3ms ;
                                    prepulse = TICK0o2ms ;
                              end
                              else begin  //error
                                    necpoj = 1'b0 ;
                                    timeout =TICK11ms ;
                                    prepulse = TICK8ms ;
                              end
                        end

                        //start of actual data (address, inverse address, data,
inverse data )
                        else if ( ( necpoj > 2 ) && ( necpoj < 70 ) ) begin
                              necpoj = necpoj + 1'b1 ; //incrementing @every edge

                              //every odd edge
                              if (necpoj[0]) begin
                                    rxbuffer = rxbuffer << 1'b1 ; //shift the
receiver buffer by 1

                                    //check whether data bit is logic 1 or zero
                                    if (  elapsed_time > 32'd1000 ) begin
                                          rxbuffer[0] = 1'b1;
                                          //elapsed_out = elapsed_time ;
                                    end
                                    else begin
                                          rxbuffer[0] = 1'b0;
                                          //elapsed_out = elapsed_time ;
```

```
                                    end
                                end

                                if ( necpoj > 66 ) begin //if we've reached all the
final burst (66th edge)
                                    if ( !( rxbuffer[31:24] & rxbuffer[23:16] ) &&
( !( rxbuffer[15:8] & rxbuffer[7:0] ) ) ) begin // if data is valid
                                        data_buff= rxbuffer[15:8] ;
                                        address_buff = rxbuffer [31:24] ;
                                    end
                                    else begin //wrong data
                                        data = 1'b0 ;
                                    end

                                    rxbuffer = 1'b0 ; //reset buffer
                                    necpoj = 1'b0 ;   //reset position
                                    timeout = TICK11ms ;
                                    prepulse = TICK8ms ;

                                    //from [lsb,msb] to [msb,lsb]
                                    for (int i = 0 ; i <= 7 ; i++) begin
                                        data[i] = data_buff[7-i] ;
                                        address[i] = address_buff[7-i] ;
                                    end
                                end
                            end

                            //error
                            else begin
                                necpoj = 1'b0 ;
                                timeout =TICK11ms ;
                                prepulse =TICK8ms ;
                            end
                        end

                        //check whether the 9ms has started
                        else begin
                            if (ir) // 9ms hasn't started
                                necpoj = 1'b0 ;
                            else
                                necpoj = 1'b1 ; //leading pulse has started for 9ms

                            timeout = TICK11ms ;
                            prepulse = TICK8ms ;
                        end
                        reset = 1'b1 ; //reset timer
                    end
                    last = {last[0],ir} ; //this is to check if there's a transition

            end


endmodule
```

A.7 decode7.sv

The decode7 module accepts the data extracted from the nec_decode module and encodes the value to the 7 segment display.

```
//7 segment decoder
```

```
//input is 4 bit
//output is active low 7 segment encoded
module decode7 (input logic [3:0] num,
            output logic [7:0] data) ;
      always_comb begin
            unique case (num)
                  0: data = 8'b1100_0000 ;
                  1: data = 8'b1111_1001 ;
                  2: data = 8'b1010_0100 ;
                  3: data = 8'b1011_0000 ;
                  4: data = 8'b1001_1001 ;
                  5: data = 8'b1001_0010 ;
                  6: data = 8'b1000_0010 ;
                  7: data = 8'b1111_1000 ;
                  8: data = 8'b1000_0000 ;
                  9: data = 8'b1001_0000 ;
                  10: data = 8'b1000_1000 ;
                  11: data = 8'b1000_0011 ;
                  12: data = 8'b1100_0110 ;
                  13: data = 8'b1010_0001 ;
                  14: data = 8'b1000_0110 ;
                  15: data = 8'b1000_1110 ;
                 default: data = 8'b1111_1111 ;
            endcase
      end

endmodule
```

A.8 timer_tb.sv

The timer_tb test bench module checks the output of timer module.

```
//timer test bench
module timer_tb ;

logic reset, clk_50m = '1 ;
logic [31:0] counter, count_test = '0 ;

timer timer_0(.CLOCK_50(clk_50m), .*) ;

initial begin

      //assert reset
      reset = '1 ;
      repeat(100)@( posedge clk_50m ) ; //100 us reset
      reset = '1 ;

      //delay for 100us
      #100ms ;

      $stop ;
end

//clock
always
      #10ns clk_50m = ~clk_50m ;

endmodule
```

A.9 nec_decode_tb.sv

The receiver_decode_tb test bench module confirms that the nec_decode module has the correct address and data received.

```
//receiver module testbench


module receiver_decode_tb ;

	logic reset, ir, CLOCK_50 = '1;
	logic [7:0] address, data ;
	logic [31:0] counter, elapsed_time, elapsed_out;

	//instantiation
	timer timer_1 (.* ) ;
	nec_decode decode_1(.elapsed_time(counter),.*) ;

	initial begin

		CLOCK_50 = 0 ;

		//assert
		ir = '1 ;
		#5ms;

		//9ms burst start
		ir = '0 ;
		#9.5ms ;

		//4.5ms space
		ir = '1 ;
		#4.5ms ;

		//lsb address - 8'b1000_0000
		for (int i = 0 ; i < 6 ; i++ ) begin
			ir = '0 ;
			#562.5us ;
			ir = '1 ;
			#562.5us ;
		end
		for (int i = 0 ; i < 2 ; i++ ) begin
			ir = '0 ;
			#562.5us ;
			ir = '1 ;
			#1125us  ;
		end

		//inverse lsb address - 8'b1111_1110
		for (int i = 0 ; i < 6 ; i++ ) begin
			ir = '0 ;
			#562.5us ;
			ir = '1 ;
			#1125us ;
		end
		for (int i = 0 ; i < 2 ; i++ ) begin
			ir = '0 ;
			#562.5us ;
			ir = '1 ;
			#562.5us ;
```

```
            end

            //lsb data - 8'b1000_0000
            for (int i = 0 ; i < 6 ; i++ ) begin
                  ir = '0 ;
                  #562.5us ;
                  ir = '1 ;
                  #562.5us ;
            end
            for (int i = 0 ; i < 2 ; i++ ) begin
                  ir = '0 ;
                  #562.5us ;
                  ir = '1 ;
                  #1125us  ;
            end

            //inverse lsb address - 8'b1111_1110
            for (int i = 0 ; i < 6 ; i++ ) begin
                  ir = '0 ;
                  #562.5us ;
                  ir = '1 ;
                  #1125us ;
            end
            for (int i = 0 ; i < 2 ; i++ ) begin
                  ir = '0 ;
                  #562.5us ;
                  ir = '1 ;
                  #562.5us ;
            end
            //last burst
            ir = '0 ;
            #562.5us ;
            ir = '1 ;
            #9.5ms ;

            #1ms ;
            $stop ;
      end

      //clock
      always
      #10ns CLOCK_50 = ~CLOCK_50 ;

endmodule
```

A.10 NECprotocol_tb.sv

The test bench NECprotocoltest module confirms that the signal transmitted from the NECprotocol module is correct.

```
module NECprotocoltest;
      logic [7:0] data, address;
      logic [31:0] counter;
      logic kphit, reset, clk_38, CLOCK_50;
      logic Tx;

      NECprotocol NEC_t (.*);
      timer time_0(.*);

      initial
```

```
	begin
		clk_38 = 1;
		CLOCK_50 = 1;
		address = 8'b0101_0101;
		data = 8'b0010_1001;
		kphit = 0;
		#1ms;
		kphit = 1;
		#1ms;
		kphit = 0;
		#70ms;

		$stop;

	end

	always
		#10ns CLOCK_50 = ~CLOCK_50;


	always
		#6600ns clk_38 = ~clk_38;
endmodule
```